Comparison of Time Metrics in Programming

Leinonen, Juho

ACM
2017-08-14

Leinonen , J , Leppänen , L , Ihantola , P & Hellas , A 2017 , Comparison of Time Metrics in Programming . in ICER '17: Proceedings of the 2017 ACM Conference on International Computing Education Research . ACM , New York , pp. 200-208 , ACM International Computing Education Research , Tacoma , Washington , United States , 18/08/2017 . https://doi.org/10.1145/310572

http://hdl.handle.net/10138/308329
https://doi.org/10.1145/3105726.3106181

acceptedVersion

Comparison of Time Metrics in Programming

Juho Leinonen
University of Helsinki
Helsinki, Finland
juho.leinonen@helsinki.fi

Leo Leppänen
University of Helsinki
Helsinki, Finland
leo.leppanen@helsinki.fi

Petri Ihantola
Tampere University of Technology
Tampere, Finland
petri.ihantola@tut.fi

Arto Hellas
University of Helsinki
Helsinki, Finland
arto.hellas@cs.helsinki.fi

ABSTRACT

Research on the indicators of student performance in introductory programming courses has traditionally focused on individual metrics and specific behaviors. These metrics include the amount of time and the quantity of steps such as code compilations, the number of completed assignments, and metrics that one cannot acquire from a programming environment. However, the differences in the predictive powers of different metrics and the cross-metric correlations are unclear, and thus there is no generally preferred metric of choice for examining time on task or effort in programming.

In this work, we contribute to the stream of research on student time on task indicators through the analysis of a multi-source dataset that contains information about students' use of a programming environment, their use of the learning material as well as self-reported data on the amount of time that the students invested in the course and per-assignment perceptions on workload, educational value and difficulty. We compare and contrast metrics from the dataset with course performance. Our results indicate that traditionally used metrics from the same data source tend to form clusters that are highly correlated with each other, but correlate poorly with metrics from other data sources. Thus, researchers should utilize multiple data sources to gain a more accurate picture of students' learning.

1 INTRODUCTION

The amount of practice it takes to become an expert has intrigued researchers for decades. General rules, such as the 10-year rule [11, 32] and the 10,000-hour rule [11, 14, 28], have been developed to estimate how laborious it is to master a skill. The rules have been fine-tuned along the way, for example by only taking deliberate practice [11] into account. While more recent research [24] has somewhat criticized these rules that promise mastery within a fixed-time period, there is no denying that the use of time on the task must have at least some kind of an effect on learning the task – one cannot become a master without practice.

A rising field within computer science education research is research based on logs of the students' programming process [17]. These logs can be very fine-grained, including information even on single keystrokes the students type while completing course activities [35]. One of the main advantages of fine-grained programming logs is the wide range of research that can be conducted with such data. For example, fine-grained log data can be aggregated into complex formats such as typing profiles [20], used to study the programming behavior of students [10], and estimate the time and effort students spend on assignments [33, 34]. In practice, data with finer granularity provides, among other things, information on how the students have reached a solution instead of just showing what the students' solutions are – this information can be used to even determine if the students have collaborated during the process [16].

The metrics that can be derived from programming logs have been used to detect struggling students in need of an intervention [1, 26]. The total time students spend on programming assignments has been found to correlate with course scores by Munson [26]. However, as programming logs only include indirect information on time, estimating the time that students actually use on an assignment is challenging [27]. For example, Munson [26] calculated the time between first and last compilations of an assignment to estimate the time that students spent on a single exercise. Murphy et al. [27] assumed that all compile events within 30 minutes of each other belong to the same programming session while Toll et al. [34] allowed up to 15-minute breaks within a single session. Additionally, both Murphy et al. [27] and Munson [26] assume that students are working on the assignment and not engaging in off-task behavior in-between compilations in a single session.

However, when only programming logs are considered, it is impossible to know what students do in-between compile events. There are many types of off-task behavior that the students might engage in – some that should be counted as practice towards the course such as studying the course material, and some that probably should not such as browsing social media. This means that additional information aside from programming logs should be considered. Even then, it is hard to know what type of metrics would be most suitable for measuring time. For example, if there are two time-like metrics available, even if they appear quite similar, there could be great differences in what they measure, which means that

ICER '17, August 18-20, 2017, Tacoma, WA, USA

ACM ISBN 978-1-4503-4968-0/17/08…$15.00
https://doi.org/10.1145/3105726.3106181

the results of a study could be totally different depending on the chosen metric.

In this work, we study time and time-like metrics derived from three separate data sources: programming logs, online material usage logs, and questionnaire answers. We look into how the metrics correlate with each other at three granularities: within a single assignment, within a single course component, and over the whole course. We are interested in learning whether some of the studied metrics could be replaced by other metrics, i.e. whether they have a strong correlation.

This article continues as follows. In Section 2 we go over some of the relevant background literature. In Section 3 we lay out the design of this research, first listing our research questions (Section 3.1), then describing the context of the study (Section 3.2), where the data for it is from (Section 3.3) and the methods we use to answer the research questions (Section 3.4). Sections 4 and 5 describe and discuss our results, respectively, with Section 5.1 describing the limitations of this work. Finally, Section 6 draws final conclusions from the results of this study.

2 BACKGROUND

2.1 Time on Task and Learning

Time on task is a term from pedagogy that refers to the amount of time that is spent on learning related activities. This period of time – as it is by definition spent actively on learning – is considered to be one of the most important factors to learning.

Tracing the origins of the notion is challenging. One of the earlier notions of the phenomenon comes from Ebbinghaus [9], who in the late 19th century acknowledged the phenomenon and sought to understand it more deeply by asking if the relationship was linear. In a series of memory experiments that involved memorizing nonsense syllable sequences, he observed that the time spent on memorizing the sequences did indeed have a near-linear relationship with the amount of remembered syllables. In another experiment where the syllable sequences were split into subsequences that were first learned and then combined, he observed that the total time spent on learning the whole task was not reduced. This led to the formation of the *total time hypothesis*, which states that a fixed amount of time is necessary to learn a fixed amount of material no matter how the task is divided.

Ebbinghaus was also one of the first to study the benefits of distributed practice versus massed practice. He studied the memorization task over a series of days, where the memorized syllable series was repeated a changing number of times, and observed that learning was most effective when distributed. That is, the total time spent on memorizing a series of syllables was longer if the memorization process was crammed together when compared to distributing the practice over multiple days.

Whilst subsequent research has shown the benefits of time on task and spaced practice over massed practice over and over again [2, 6], students' decisions on how to study are influenced by numerous competing factors. As a consequence, massed practice is often preferred over spaced practice, even when students have explicitly been given feedback on their better performance with spaced practice [18].

This observation on some preferring to use non-optimal learning practices lends directly to the studies conducted by Ericsson et al. [11]. They studied the practice of expert violinists, and noted that the high-performing individuals had a habit of deliberately focusing on the areas that they were lacking in instead of simply practicing the songs over and over again. They suggested that in order to truly excel at something, one must "step outside the comfort zone" and deliberately practice the challenging activities over and over again – something that many choose not to do.

Whilst Ericsson observed that developing expertise takes years, neither Ebbinghaus nor Ericsson claimed that each individual would learn at the same pace. The observation that the speed of learning varies among individuals [4, 37] has led to the development of teaching approaches that take this variety into account, including the Mastery Learning approach [3], where students are expected to master the current tasks before they are allowed to advance to the next tasks.

2.2 Factors Affecting Performance in Programming Courses

Factors affecting students' performance in programming courses have traditionally been studied with the purpose of being able to predict the students' performance. In this line of research, a myriad of predictors ranging from students' affective states [31], students' programming behavior [8, 10, 12, 13, 21, 27, 38] to complex programming process based metrics [5, 20, 30] have been constructed. Research has also been invested in analyzing and reviewing different metrics for student performance [39]. Much of the research on performance in programming courses has been aimed at identifying at-risk students early enough to intervene and help the students learn the course contents and thus pass the course [1, 26].

Murphy et al. [27] have developed a tool called Retina to give both students and instructors a better idea of how students are performing on the course. The tool offers information on errors made on exercises, but also information on how much time it takes on average to complete each exercise. They note that it is practically impossible to know the exact amount of time that students work on assignments solely based on programming log data.

Several studies have highlighted time as an integral factor in course success. For example, Watson et al. [39] found that in addition to an metric constructed from sequential compilation events [38], the percentage of time that students spent on resolving errors in a programming lab was indicative of their future performance in the course. They studied 38 traditional and 12 new metrics of student success. The new metrics were solely based on programming log data whilst the traditional metrics were based on for example student background variables such as previous programming experience and questionnaire answers.

An emerging field of detecting at-risk students is using machine learning methods instead of relying on the educator for noticing struggling students [1, 26]. Recently, Munson [26] studied automated metrics for assessing novice programmers' performance early enough in the course for an intervention. Munson derived numerous metrics from programming logs and analyzed their correlations with course scores and each other. The metrics included session time, error, edit, and compile related measurements. He

found that especially total session time, i.e. the amount of time the student spent programming, had a moderate positive correlation with course scores. Additionally, the amount of changes to the source code had a similar positive correlation with course scores. Interestingly, total session time and the amount of change events were highly correlated, which could indicate that the amount of changes to the source code is a good metric for time. However, as partial correlations between the metrics were not studied, it remains possible that the correlation could be at least partially explained by other variables that correlate with both the amount of changes and total session time.

As noted by Ihantola et al. [17], it is questionable whether specialized metrics generalize to other contexts as many studies are conducted within a single course at a single institution with custom metrics, which makes it hard to replicate such studies. Even with an increasing amount of research conducted based on time and effort metrics derived from programming logs, there are no standardized metrics for measuring time based on these logs. It is also unclear how such metrics are related to each other.

3 METHODOLOGY

3.1 Research Questions

The overview of the background literature in Section 2 paints a picture where a large amount of research has focused on two types of predictors of student success. Some predictors are essentially metrics of "quality" such as the amount of errors in program code, whilst other predictors are essentially metrics of time and effort in that they measure how much work or time or effort the student put into studying, for example by calculating the amount of changes to the source code or estimating the time spent based on timestamps.

This raises a question: how are these different time-like metrics related to each other? To allow for a better understanding and comparison of results from different studies, we take a look at correlations between a certain group of time-like metrics available to us and answer the following research questions:

- RQ1: How do common time and time related metrics correlate on a per-assignment basis?
- RQ2: How do common time and time related metrics correlate within larger course components?
- RQ3: How do common time and time related metrics correlate over the whole course?

3.2 Context of the Study

The data for this study has been gathered from two introductory programming courses organized at University of Helsinki, a research-oriented university in Europe. The courses were held during spring and fall of 2016 and one of the authors of this paper is also responsible for organizing both courses. The programming language that is taught in the course is Java, and the contents of the course are similar to many other introductory programming courses offered at universities: variables, input/output, selection, objects, lists, sorting, and searching. The courses lasted for seven weeks each.

The teaching method in the course expects that the majority of the time in the course is spent on working on programming assignments. To provide support, the computer science department offers open labs with a total of 70 computer seats and additional places for those with laptops. Instructors and teaching assistants attend the open labs providing support (for additional details, see [19]).

The course is mandatory for students majoring in computer science, and they are expected to take it in the first teaching period of their first year. Other students can take it if they feel that it would benefit their studies, or if they are considering computer science as a potential minor subject. Non-CS students often take the course later in their studies, for example in the second year. During spring 2016, the course was graded using a pen-and-paper exam and a computer-based exam, and during fall 2016, the course was graded using three take-home exams. The course grading schemes were slightly different, but approximately 40% of the overall course mark comes from the exams, and 60% comes from working on sets of individual programming assignments and pair-programming assignments.

3.3 Data Sources and Variables

During the courses, participants provided data through many avenues. The students used the NetBeans IDE with the Test My Code (TMC) -plugin [36] as they worked on the programming assignments. The plugin recorded the students' programming process and automatically assessed the correctness of students' submissions. For each submitted assignment that was solved correctly, the students were asked to rate the difficulty, workload, and educational value of the assignment. Additionally, the online learning material that the students used stored details on their use of the material, and finally, questionnaires were administered to gather data regarding the students' weekly use of time in the course.

3.3.1 Working Environment and Assessment Server. The NetBeans IDE with TMC provides the functionality needed to download and submit programming assignments, as well as the typical programming environment functionality such as running and testing the code that one is working on. The students were able to run assignment specific test suites that gave them feedback on what parts of the assignment were correctly implemented and provided hints towards solving some of the simpler and more common mistakes.

The environment was augmented to record and store the students' working process data on each programming project that was related to a programming assignment on the course. The data includes timestamps for every modification that the students do on the programming assignment templates as well as project specific actions such as running the code, testing the code and submitting the project. Whenever the student submitted an assignment to the server, a set of unit tests was run on the assignment. Once the unit tests were executed, feedback on the correctness of the student's solution so far was provided back to the student. Multiple submissions were allowed and there was no penalty associated with submitting incomplete or incorrect solutions.

The programming process data was further analyzed by extracting assignment specific information that contains (1) total time spent on each assignment, (2) total active time spent on each assignment, (3) number of code edit events, (4) number of paste events, (5) number of times that the assignment was run, (6) number of times that the assignment was tested, (7) number of times that the assignment was submitted, (8) number of times the students used the debug functionality of the IDE, (9) the number of times the IDE

either gained or lost focus, (10) sum of counts 3 through 9, and (11) the average typing speed of the student.

The total time spent on the assignment was calculated based on the first event and the last event where the student modified the source code similar to Munson's work [26]. The total active time was the same, but excluding pauses longer than 3 minutes. The average typing speed was calculated by extracting all intervals between edit events (that is, individual key presses) between 10 and 750 milliseconds in length similar to Longi et al. [23].

3.3.2 Online Course Material Usage Data. The course used an online ebook with embedded assignment descriptions. The ebook was divided into seven course components, each containing theory, program snippets, worked examples and assignment descriptions for the specific week of the course. The course components were long HTML pages; each page can be considered an analogue to a chapter of a traditional textbook. The online ebook used a JavaScript library that stores information on the use of the materials [22].

The stored data includes information on each time when a user scrolls the page, or stays in the same location of the page for a predefined time interval or a "tick", and can be used to measure the amount of time a user spends at different parts of the online material. If the user spent more than three minutes on the same exact location without moving, the JavaScript library stopped storing the events until the next time that the user became active.

Weekly information on the amount of scroll events and ticks on the material was extracted for the analysis.

3.3.3 Questionnaire Data. After completing an assignment, the students were prompted for information on the assignment. The per-submission questionnaires contained three specific questions, one each regarding the educational value, difficulty and workload of the assignment. Answers to each of these were provided on a 5-step Likert-scale with 1 indicating "not at all" and 5 "extremely". The participants were also able to provide feedback in a free text form, but these textual answers were not analyzed as a part of this study. The students were not required to answer these questionnaires and were not incentivized to answer them.

Furthermore, at the end of each week, the students were asked to provide a self-estimate of the time they had spent working on the course during that week.

3.3.4 Course Exam. During spring 2016, the course had both a pen-and-paper exam as well as a computer exam, and during fall 2016, three computer exams were administered. As the courses had a different number of exams, administered using different means and with different questions, we use only the combined exam scores for both courses as variables in this study. That is, each student has a single variable that contains how many points the student obtained overall in all the assignments of all the exams of the respective course.

3.3.5 Summary. A summarizing listing of the variables used in this study – grouped by source – is provided as Table 1.

Overall, 406 students participated in the courses in total. After excluding the students who chose to opt out from the study, who had participated in the same programming course previously, and who did not answer at least a single self-report questionnaire, a total of 309 students was left for the analysis.

Table 1: A listing of the variables and their sources used in the study

Source	Variable
Process data	Time spent on each assignment
Process data	Active time spent on each assignment
Process data	Edit event count
Process data	Paste event count
Process data	Run event count
Process data	Test event count
Process data	Submit event count
Process data	Debug event count
Process data	Focus changes
Process data	Event count
Process data	Average typing speed
Assessment	Assignment correctness
Assessment	Points from assignments
Material	Weekly scroll event count
Material	Weekly tick event count
Exams	Total exam points
Questionnaire	Assignment-specific perceived educational value
Questionnaire	Assignment-specific perceived difficulty
Questionnaire	Assignment-specific perceived workload
Questionnaire	Estimated time spent on course each week

3.4 Method

The data from the sources described in the previous subsection were combined into a singular data set. For each research question, the data was normalized to have 0 mean and variance of 1 within the corresponding groups: within each assignment for the first research question, within each course component for the second research question, and finally over the whole course for the third research question. The questionnaire data was left unnormalized.

In the case of the first research question, our data includes students who did not answer the questionnaires. In the data set used to answer research question two, students who did not answer the questionnaires were excluded.

From these normalized data sets, correlations were calculated between all pairs of variables. After that, correlation matrices were reordered by using hierarchical clustering[1] to facilitate visual analysis of the results. Finally, partial correlations between all variable pairs were estimated from the previously calculated correlation matrix[2] so that the effects of all other variables were excluded.

We use Spearman's rank correlation coefficient to measure relatedness between our variables since some of the correlations are nonlinear. Unlike Pearson's correlation coefficient, which measures *linear* relation, Spearman's rank correlation coefficient measures how well the relation between the two variables is explainable by a *monotonic* function [15]. As an added benefit, Spearman's rank correlation coefficient does not require the variables to be normally distributed and is as such more resilient towards outliers [25].

All calculated p-values were corrected for multiple comparisons using the Bonferroni correction [7], which controls the familywise

[1] https://cran.r-project.org/web/packages/corrplot/
[2] https://cran.r-project.org/web/packages/corpcor/

error rate of multiple comparisons. The correction simply modifies the threshold of statistical significance from $\alpha = 0.05$ to α/n, where n is the number of comparisons made. Essentially, the Bonferroni correction is used to avoid finding correlations based on random chance due to the large amount of pairwise correlations that are studied in this work.

4 RESULTS

Our results are presented visually in Figures 1–6. Non-significant correlations in Figures 1, 3 and 5 are excluded and shown as crosses, other correlations are statistically significant after Bonferroni correction. The size of the diagonal correlations (correlation of the variable with itself, i.e. $r = 1$) can be used for visually estimating the strength of the other correlations.

4.1 Per-Assignment Results

In order to answer the first research question, "How do common time and time related metrics correlate on a per-assignment basis?", we observe the pairwise correlations plotted in Figures 1 and 2. These correspond to students' effort during individual assignments.

Figure 1 presents the full correlations between all pairs of variables in the data. We note that the graph contains multiple clusters wherein the variables are highly correlated with each other. The largest of these is the cluster in the top-left corner of the graph, where active work time, total count of source code events, count of code edit events, number of times the program was run and number of focus change events are all correlated with each other positively (correlations range between $r = .4$ and $r = .9$). Specifically, active time correlates with total count of source code events ($r = .79$), edit event count ($r = .72$), run event count ($r = .53$) and focus change event count ($r = .54$). These same variables are then mildly correlated with student perceptions of educational value, workload and difficulty, as well as number of paste events (these correlations range between $r = .15$ and $r = .3$).

Next up, the number of times the debug feature was used is correlated positively with the total time (including pauses) used on the assignment ($r = .4$). Both of these are then negatively correlated with maximum assignment correctness ($r = -.25, r = -.35$). Finally, a cluster of positive correlations forms between student perceptions of educational value, workload and difficulty (these correlations range between $r = .5$ and $r = .8$).

Observing the pairwise partial correlations where all other variables have been used as control variables (see Figure 2), we notice essentially the same clusters, albeit with significantly weaker correlations. The total event count and code edit counts are very strongly correlated ($r = .9$), as are student perceptions of difficulty and workload ($r = .7$). We note that typing speed is unsurprisingly negatively correlated with total time spent ($r = -.3$), as is the count of paste events with the count of code edit events ($r = -.3$). The negative correlation between assignment correctness and total time is essentially unaffected compared to the full correlations.

One additional change from the full correlations is the new negative correlation between number of edits and number of focus changes ($r = -.3$).

4.2 Per-Week Results

For the second research question, "*How do common time and time related metrics correlate within larger course components?*" we observe the pairwise correlations plotted in Figures 3 and 4. These correspond to students' effort within single weeks of the studied course.

The full correlations in Figure 3 are largely similar to those presented in Figure 1. Similar clusters form in both between largely the same variables. Perhaps the clearest difference is that the negative correlations with assignment correctness disappear when inspecting a course week instead of single assignments. Instead, we see a new medium-strength correlation between assignment correctness and edit count ($r = .41$), submission count ($r = .38$), event count ($r = .35$) and active time ($r = .34$). We furthermore note that both scroll and tick counts retrieved from material usage metrics ($r = .73$) are correlated with the larger cluster of programming environment count (these correlations range between $r = .3$ and $r = .5$). An additional value representing students' self-reported weekly time on the course is also included – the correlation between the self-reported time and active time is $r = .5$.

Continuing to the partial correlations presented in Figure 4, we note that it, too, is very similar to Figure 2. We note that the only negative correlation of any significance is between the number of edits and the number of focus changes ($r = -.33$). This same correlation was present in the per-assignment partial correlations. Compared to the full correlations, the correlations between the learning material event counts and the programming environment event counts essentially disappear when partial correlations are considered, as do the correlations with the students' self-reported total time.

4.3 Full Course Results

For the third research question, "*How do common time and time related metrics correlate over the whole course?*", we observe Figures 5 and 6.

Figure 5 presents the full correlations over the combined data set that includes both courses. The plot contains two larger striking features: first of all, almost everything is correlated with almost everything else. Secondly, the only outliers regarding that are total time spent – which does not correlate with exam scores or assignment correctness – and exam scores. Exam scores show a relatively strong correlation with assignment correctness ($r = .6$), which is easily understandable. Interestingly, while exam scores do correlate positively with active time ($r = .2$), the correlation with total time is not statistically significant when corrected.

Figure 6 shows the partial correlations between all variable pairs over the complete courses. These results seem to be largely in line with those presented in relation to the other partial correlation plots. We note that the number of negatively correlated pairs is, however, slightly larger than previously. Assignment correctness is negatively correlated with the programming environment event count ($r = -.24$). The number of source code edits is also negatively correlated with the number of focus changes to and from the programming environment ($r = -.25$) and the number of paste events ($r = -.25$). Peculiarly, the time spent in the course ebook

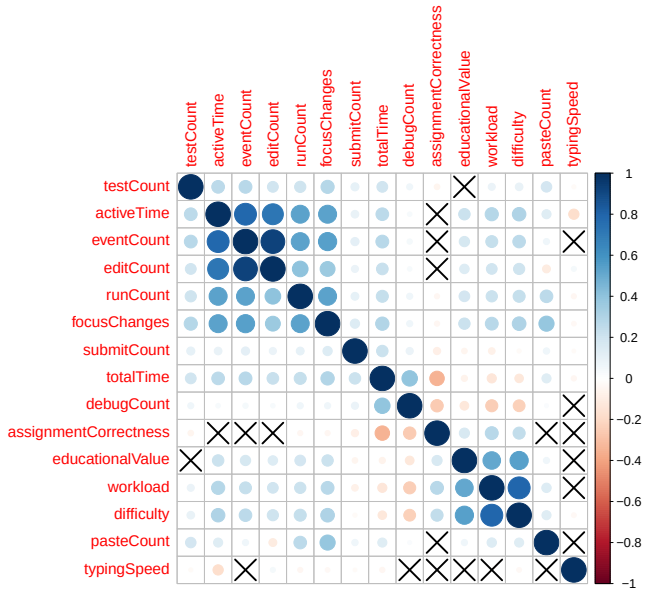

Figure 1: Per-assignment Spearman-correlations with significances corrected using the Bonferroni correction for multiple comparisons. Size and color of circles indicate the Spearman correlation coefficient. Crosses indicate that the correlation is not statistically significant.

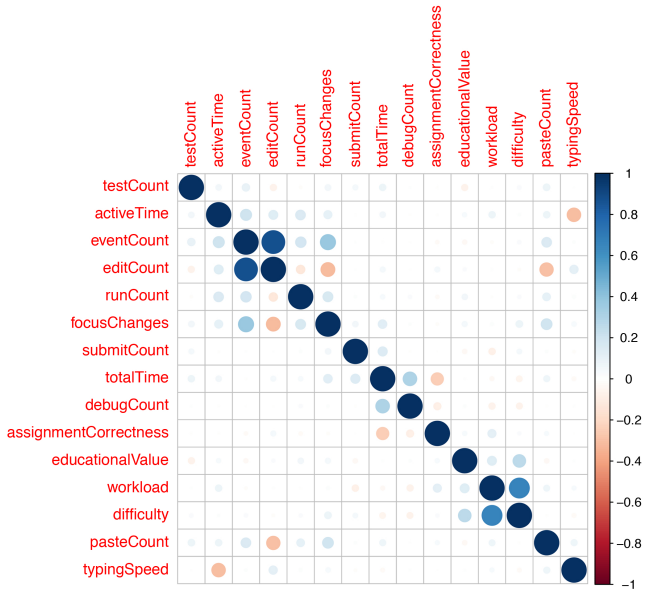

Figure 2: Pairwise per assignment partial Spearman-correlations controlled for all the other variables. Size and color of circles indicate the Spearman correlation coefficient.

(tick count) is very slightly negatively correlated with total time spent on assignments ($r = -.14$).

In the partial correlations, the exam scores are positively correlated with assignment correctness ($r = .44$), and slightly correlated with the number of paste events ($r = .19$). They also show a very slight negative correlation with total time ($r = -.12$), but this correlation was not statistically significant as a full correlation.

5 DISCUSSION

Our results show that there is a large amount of correlation within certain variable clusters. Events recorded within the programming environment tend to correlate, as do events recorded from the learning material. Due to this, it is important to look at partial correlations of the variables. Essentially, partial correlations are used to study whether any two variables correlate when controlling for other variables in the data set.

The largest cluster of highly-correlating variables we observed consisted of variables collected from the IDE the students used. First of all, active programming time is highly correlated with the number of actions taken in the programming environment and the number of code edits. It is also relatively well correlated with the count of program runs and changes of focus either out of or to the programming environment. These together form a large cluster of variables that are well-correlated with each other.

At the same time, this first cluster is only slightly correlated with the counts of test runs, debug usage, total time (including pauses), number of paste events and the number of times the code is submitted. There is also very little correlation between this cluster of variables and the perceived amounts of educational value, workload or difficulty for each assignment.

When partial correlations are considered, this cluster essentially reduces to a strong correlation between the total number of events and code edits. This indicates that most of the actions taken by the students in the programming environment are edits to code.

While these relations – as well as the negative correlation between the typing speed and the amount of active time – are somewhat self-evident they do suggest that the analytical methods are correct. Thus, these "trivial" results support the existence of the more curious phenomena we observe next.

One such curious phenomenon is the negative correlation between the numbers of edits and focus changes that is also visible when corrected for the other variables. This indicates that the students who do more changes to their code tend to not change in and out of their editor as much as students with less changes to their code. One possible explanation for this is that some students tend to tinker their code in the editor if they are having trouble whereas others go to the material for help [29].

Also curious is the behavior of the number of debug events: it is negatively correlated with assignment correctness, perceived educational value, workload and difficulty. While the first one is explainable by the fact that struggling students are likely to both use the debug tool and abandon an assignment before finishing it completely, the negative correlations with perceptions regarding

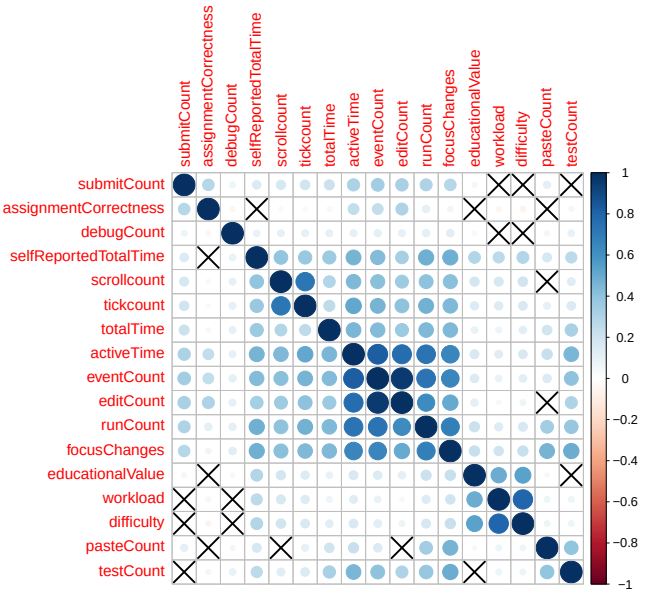

Figure 3: Within week, i.e. per course component Spearman-correlations with significances corrected using the Bonferroni correction for multiple comparisons. Size and color of circles indicate the Spearman correlation coefficient. Crosses indicate that the correlation is not statistically significant.

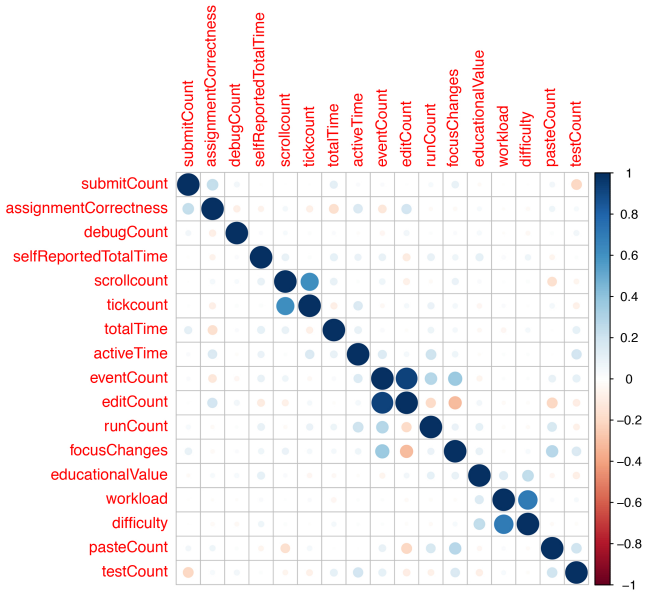

Figure 4: Pairwise within week, i.e. per course component partial Spearman-correlations between the tested variables controlled for all the other variables. Size and color of circles indicate the Spearman correlation coefficient.

the assignment are not as intuitive. One possible explanation is that the debug feature is mostly used by advanced students, or in other words, students who do not struggle with the assignments and want to understand the functionality of the programs better.

We further note that while perceived educational value, workload and difficulty are correlated in the full correlation analysis, partial correlation analysis indicates that perceived educational value is only very slightly correlated with the others when partial correlations are observed. In other words, assignments that are difficult or laborious are not necessarily beneficial for learning, and beneficial assignments do not need to be laborious or difficult. Additionally, these self-reported metrics, including the self-reported hours spent each week, did not have significant correlations with other metrics when examining the partial correlations, which raises questions about the validity of self-reported metrics altogether.

We failed to find any significant partial correlations between metrics obtained from the ebook-like learning material and the programming environment. Our belief is that this is partially due to focusing on too coarse grained data – it is likely that different students spend different proportions of time in different learning environments and material locations. Thus, effort should not be estimated solely based on the total study time, as even small pauses can account for large variations in learning outcomes [21]. Moreover, it is also important to know what the time is spent on – for example, knowledge on which material paragraphs the students spend their time on could provide additional insights on the students' struggles [22].

Interestingly, essentially none of the variables are strongly correlated with exam scores: exam scores are most strongly correlated with assignment correctness and slightly correlated with paste counts, number of assignment submissions, active programming time, number of programming events and code edit event counts. When partial correlations are considered, only assignment correctness and paste counts are correlated in any significant magnitude with exam scores. Based on this, it seems that in our context, it would not be sensible to conduct interventions based on time related metrics alone without additional proof of the student needing help – however, it is also possible that the change from the pen-and-paper -based exam to the computer exam has influenced the overall exam data.

5.1 Limitations

One of the core limitations of this work is that the data comes from a type of a university course with high attrition rates. In our context where over 400 students were initially active, data from marginally over 300 was used. We excluded information from students who chose to not participate in the study, who did not answer any of the questionnaires, and who potentially had e.g. JavaScript blocking scripts on their computers (students with practically no data from the material logging component). This means that there is a risk of selection bias. However, we tried to combat these problems by normalizing the data whenever possible so that the metrics would be comparable as well as taking averages instead of summing so that missing data points (e.g. a student not answering a questionnaire) would not affect the results as much.

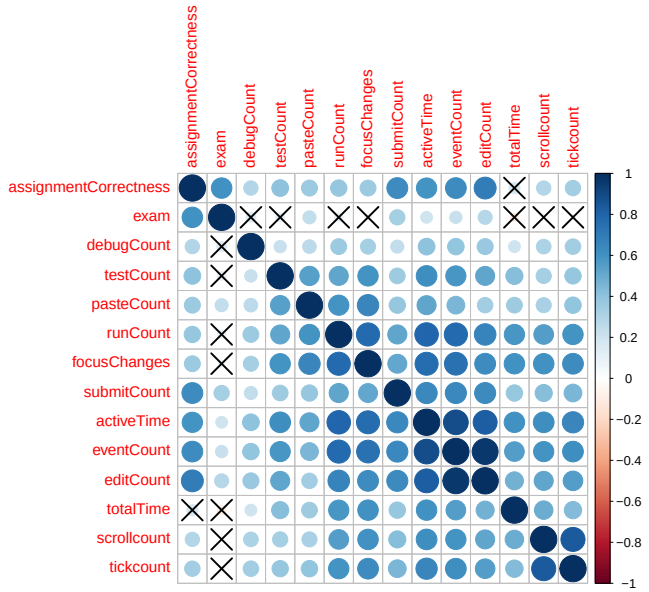

Figure 5: Full course Spearman-correlations with significances corrected using the Bonferroni correction for multiple comparisons. Size and color of circles indicate the Spearman correlation coefficient. Crosses indicate that the correlation is not statistically significant.

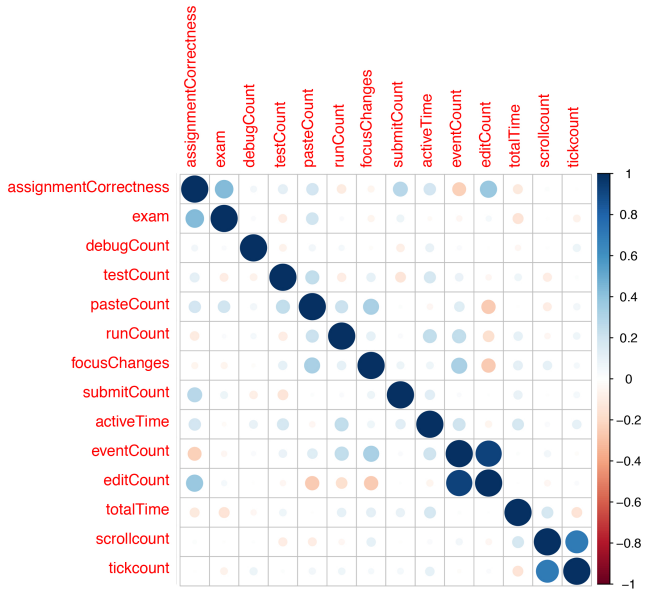

Figure 6: Full course partial pairwise Spearman-correlations between the tested variables controlled for all the other variables. Size and color of circles indicate the Spearman correlation coefficient.

A concern for the external validity and generalizability of the study arises from the fact that all the data used in this study comes from a single university. We sought to combat this by including data from multiple course instances, but acknowledge that the course content is the same. We encourage fellow researchers to replicate our study in their context, and are willing to provide support and the necessary tools to do so.

Additionally, the course material usage data might differ depending on the specifications of the resolution of the screen of the user as the JavaScript library used to study material usage tracks things visible on the user's screen. A larger screen with a higher resolution can have more content on the screen at the same time. Thus, users with low-resolution monitors might have more scroll events compared to users with high-resolution screens.

6 CONCLUSIONS

In this work, we looked at different time metrics from multiple different data sources. We calculated the correlations between the metrics within single assignments, single thematically coherent course components, and over the whole course. We found that many of the metrics form clusters indicating possible redundancy. Further analysis of partial correlations revealed that some metrics are indeed most likely redundant, for example students' self-reported workload and difficulty.

With this work, we have sought to bring attention to the myriad of variables that can potentially be used to measure students' activity and time usage in the course. If this multitude of factors is not taken into account, the best case scenario is that separate research streams that study these variables merely create redundancy as research groups report on distinct but highly correlated variables as all explaining some other variable. In the worst case, strands of research that in actuality describe the same underlying phenomena are not recognized as related if they use slightly different – but in actuality highly related – variables as both the explaining and explained variables.

Interestingly, we found that exam scores are not strongly correlated with any of the studied metrics. Based on our results, when corrected for independence, the self-reported educational value of an assignment does not have a strong correlation with assignment difficulty and workload, which indicates that an assignment can be educational even if it is not laborious. Additionally, we noticed that material usage metrics do not have significant correlations with metrics built from programming logs. This means that in order to get an accurate picture of students' learning, data from all the learning environments the students use should be combined.

As part of our future work, we are interested in combining student background information with the data studied here. For example, there could be differences in how well certain metrics perform depending on factors such as previous programming experience. Additionally, we are interested in building predictive models based on the time-like metrics examined in this study.

ACKNOWLEDGEMENTS

This work was partially funded by Academy of Finland under grant number 303694 *Skills, education and the future of work.*

REFERENCES

[1] Alireza Ahadi, Raymond Lister, Heikki Haapala, and Arto Vihavainen. 2015. Exploring machine learning methods to automatically identify students in need of assistance. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ACM, 121–130.

[2] AD Baddeley and DJA Longman. 1978. The influence of length and frequency of training session on the rate of learning to type. *Ergonomics* 21, 8 (1978), 627–635.

[3] Benjamin S Bloom. 1974. Time and learning. *American psychologist* 29, 9 (1974), 682.

[4] John B Carroll. 1963. A model of school learning. *Teachers college record* (1963).

[5] Adam S Carter, Christopher D Hundhausen, and Olusola Adesope. 2015. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the eleventh annual International Conference on International Computing Education Research*. ACM, 141–150.

[6] John Dunlosky, Katherine A Rawson, Elizabeth J Marsh, Mitchell J Nathan, and Daniel T Willingham. 2013. Improving students' learning with effective learning techniques: Promising directions from cognitive and educational psychology. *Psychological Science in the Public Interest* 14, 1 (2013), 4–58.

[7] Olive Jean Dunn. 1961. Multiple comparisons among means. *J. Amer. Statist. Assoc.* 56, 293 (1961), 52–64.

[8] Gregory Dyke. 2011. Which aspects of novice programmers' usage of an IDE predict learning outcomes. In *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 505–510.

[9] Herm Ebbinghaus. 1885. Ueber das Gedächtnis. (1885).

[10] Stephen H Edwards, Jason Snyder, Manuel A Pérez-Quiñones, Anthony Allevato, Dongkwan Kim, and Betsy Tretola. 2009. Comparing effective and ineffective behaviors of student programmers. In *Proceedings of the fifth international workshop on Computing education research workshop*. ACM, 3–14.

[11] K Anders Ericsson, Ralf T Krampe, and Clemens Tesch-Römer. 1993. The role of deliberate practice in the acquisition of expert performance. *Psychological review* 100, 3 (1993), 363.

[12] Anthony Estey and Yvonne Coady. 2016. Can Interaction Patterns with Supplemental Study Tools Predict Outcomes in CS1?. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 236–241.

[13] Anthony Estey, Hieke Keuning, and Yvonne Coady. 2017. Automatically Classifying Students in Need of Support by Detecting Changes in Programming Behaviour. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 189–194.

[14] Malcolm Gladwell. 2008. The 10 000 hour-rule. In *Outliers: the story of success*. Little, Brown and Company, New York, 35–68.

[15] J Hauke and T Kossowski. 2011. Comparison of values of Pearson's and Spearman's correlation coefficient on the same sets of data. *Quaestiones Geographicae* 30, 2 (2011).

[16] Arto Hellas, Juho Leinonen, and Petri Ihantola. 2017. Plagiarism in Take-home Exams: Help-seeking, Collaboration, and Systematic Cheating. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 238–243.

[17] Petri Ihantola, Arto Vihavainen, Alireza Ahadi, Matthew Butler, Jürgen Börstler, Stephen H Edwards, Essi Isohanni, Ari Korhonen, Andrew Petersen, Kelly Rivers, and others. 2015. Educational data mining and learning analytics in programming: Literature review and case studies. In *Proceedings of the 2015 ITiCSE on Working Group Reports*. ACM, 41–63.

[18] Nate Kornell and Robert A Bjork. 2007. The promise and perils of self-regulated study. *Psychonomic Bulletin & Review* 14, 2 (2007), 219–224.

[19] Jaakko Kurhila and Arto Vihavainen. 2011. Management, Structures and Tools to Scale Up Personal Advising in Large Programming Courses. In *Proceedings of the 2011 Conference on Information Technology Education (SIGITE '11)*. ACM, New York, NY, USA, 3–8. DOI: http://dx.doi.org/10.1145/2047594.2047596

[20] Juho Leinonen, Krista Longi, Arto Klami, and Arto Vihavainen. 2016. Automatic inference of programming performance and experience from typing patterns. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. ACM, 132–137.

[21] Leo Leppänen, Juho Leinonen, and Arto Hellas. 2016. Pauses and spacing in learning to program. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. ACM, 41–50.

[22] Leo Leppänen, Juho Leinonen, Petri Ihantola, and Arto Hellas. 2017. Using and collecting fine-grained usage data to improve online learning materials. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*. IEEE Press, 4–12.

[23] Krista Longi, Juho Leinonen, Henrik Nygren, Joni Salmi, Arto Klami, and Arto Vihavainen. 2015. Identification of programmers from typing patterns. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. ACM, 60–67.

[24] Brooke N Macnamara, David Z Hambrick, and Frederick L Oswald. 2014. Deliberate practice and performance in music, games, sports, education, and professions a meta-analysis. *Psychological science* 25, 8 (2014), 1608–1618.

[25] MM Mukaka. 2012. A guide to appropriate use of correlation coefficient in medical research. *Malawi Medical Journal* 24, 3 (2012), 69–71.

[26] Jonathan P Munson. 2017. Metrics for timely assessment of novice programmers. *Journal of Computing Sciences in Colleges* 32, 3 (2017), 136–148.

[27] Christian Murphy, Gail Kaiser, Kristin Loveland, and Sahar Hasan. 2009. Retina: helping students and instructors based on observed programming activities. *ACM SIGCSE Bulletin* 41, 1 (2009), 178–182.

[28] David A Omahen. 2009. The 10 000-hour rule and residency training. *Canadian Medical Association Journal* 180, 12 (2009), 1272–1272.

[29] David N Perkins, Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. 1986. Conditions of learning in novice programmers. *Journal of Educational Computing Research* 2, 1 (1986), 37–55.

[30] Andrew Petersen, Jaime Spacco, and Arto Vihavainen. 2015. An exploration of error quotient in multiple contexts. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. ACM, 77–86.

[31] Ma Mercedes T Rodrigo, Ryan S Baker, Matthew C Jadud, Anna Christine M Amarra, Thomas Dy, Maria Beatriz V Espejo-Lahoz, Sheryl Ann L Lim, Sheila AMS Pascua, Jessica O Sugay, and Emily S Tabanao. 2009. Affective and behavioral predictors of novice programmer achievement. In *ACM SIGCSE Bulletin*, Vol. 41. ACM, 156–160.

[32] Herbert Simon and William Chase. 1988. Skill in chess. In *Computer chess compendium*. Springer, 175–188.

[33] Daniel Toll. 2016. *Measuring Programming Assignment Effort*. Ph.D. Dissertation. Faculty of Technology, Linnaeus University.

[34] Daniel Toll, Tobias Olsson, Morgan Ericsson, and Anna Wingkvist. 2016. Fine-grained recording of student programming sessions to improve teaching and time estimations. In *International Journal of Engineering, Science and Innovative Technology*, Vol. 32. 1069–1077.

[35] Arto Vihavainen, Matti Luukkainen, and Petri Ihantola. 2014. Analysis of source code snapshot granularity levels. In *Proceedings of the 15th Annual Conference on Information technology education*. ACM, 21–26.

[36] Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, and Martin Pärtel. 2013. Scaffolding students' learning using test my code. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*. ACM, 117–122.

[37] Herbert J Walberg. 1988. Synthesis of research on time and learning. *Educational leadership* 45, 6 (1988), 76–85.

[38] Christopher Watson, Frederick WB Li, and Jamie L Godwin. 2013. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*. IEEE, 319–323.

[39] Christopher Watson, Frederick WB Li, and Jamie L Godwin. 2014. No tests required: comparing traditional and dynamic predictors of programming success. In *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 469–474.